Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Račič

Mobilna aplikacija za razpoznavanje objektov iz več pogledov

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: izr. prof. dr. Matej Kristan

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V nalogi obdelajte problem razpoznavanja objektov na nizkoporabnih mobilnih platformah. Za izhodišče izberite metode, ki so na področju računalniškega vida trenutno najuspešnejše. Posebno pozornost posvetite zahtevi, da morajo biti metode portabilne na mobilne sisteme, kot so pametni telefoni. Izberite ustrezno okolje za razvoj mobilne aplikacije. Aplikacija naj bo sposobna razpoznavati vnaprej izbrane objekte iz več pogledov. Izdelajte podatkovno zbirko in analizirajte metode s stališča uspešnosti razpoznavanja. Prav tako komentirajte možnost poganjanja metode za razpoznavanje na pametnem telefonu.

Moja zahvala gre mentorju izr. prof. dr. Mateju Kristanu za odzivnost, dodatno razlago in strokovno pomoč pri izdelavi diplomskega dela vse do zaključka. Zahvaljujem se mu, ker navkljub dolgotrajnemu procesu ni izgubil zaupanja vame. Prav tako se zahvaljujem tudi družini in dekletu za podporo, pomoč in spodbudo tako pri študiju kot na drugih področjih.

Kazalo

kratica	angleško	slovensko
CNN	Convolutional Neural Network	konvolucijska nevronska mreža
GPU	Graphics Processing Unit	grafična procesna enota
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge	ImageNet tekmovanje v vizualnem razpoznavanju večjega števila podatkov
HOG	Histogram of Oriented Gradients	histogram orientiranih gradientov
ReLU	Rectified Linear Units	usmerjene linearne enote
RPN	Region Proposal Network	mreža za priporočanje regij

Povzetek

Naslov: Mobilna aplikacija za razpoznavanje objektov iz več pogledov

Avtor: Matej Račič

Diplomsko delo opisuje problem klasifikacije objektov na sliki s pomočjo mobilne aplikacije. Podrobneje smo preizkusili implementacijo Inception [29], ki z uporabo okolja Tensorflow [30] omogoča enostavno spremljanje in optimizacijo učenega procesa klasifikatorja. Zajeli smo specialno zbirko za realistično objektivno analizo delovanja aplikacije. Predstavili smo dobre prakse v procesu učenja in vpliv parametrov, kot sta število iteracij in učnih primerov. Vso znanje smo uspešno uporabili in klasifikator prilagodili razpoznavanju na izbrani domeni za doseganje želenih rezultatov in uspešno klasifikacijo objektov iz različnih pogledov. S pomočjo okolja Android Studio [2] smo tudi generirani razpoznavalnik prenesli na mobilno napravo.

Ključne besede: mobilne naprave, razpoznavanje objektov, nevronske mreže.

Abstract

Title: A mobile application for multiview object recognition

Author: Matej Račič

The thesis deals with the problem of multiview object recognition with a mobile application. To achieve this, we have used the neural network architecture called Inception [29], that uses the Tensorflow [30] environment which enables simple modifications of the learning process. We used a specific collection of images, to help us objectively and realistically evaluate how well the object recognition works. We have presented good practices and we have applied that knowledge in the development of our object recognition. We have also examined various effects of different parameters such as number of iterations and training samples. We have combined it all and modified the model for recognising our own selected categories and maximising the results of the object recognition. Using Android Studio [2] we have transferred the generated model to a mobile device.

Keywords: mobile devices, object recognition, neural networks.

Poglavje 1

Uvod

1.1 Motivacija

Različni pristopi nevronskih mrež uspešno rešujejo najrazličnejše probleme, vendar se pogosto srečujejo s problemi prostorske in procesorske kompleksnosti, kot tudi s pomanjkljivimi količinami podatkov. Za razpoznavanje objektov na slikah se že nekaj časa uporabljajo konvolucijske nevronske mreže, vendar so pristopi prostorsko in časovno kompleksni tako za učenje ter tudi za razpoznavanje. Ker učenje nevronskih mrež zahteva veliko časa in procesorske moči, lahko izkoristimo metodo prenosa znanja, ki to zmanjša in na ta način pristop približa večjemu število uporabnikov. Optimizacijo na področju kompleksnosti razpoznavanja so omogočili tudi moduli Inception [29], ki zmanjšajo število parametrov in s tem tudi prostorsko kompleksnost. Klasifikacija objektov na sliki je eno izmed mnogih področij, ki jih rešujemo z uporabo umetne inteligence. Na področju računalniškega vida le-ta omogoča pospešitev, filtriranje in do neke mere avtomatizacijo opredeljevanja fotografij. Z avtomatizacijo prepoznavanja objektov na slikah hkrati omogočamo anonimnost uporabnikom in avtomatsko filtriranje nezaželenih vsebin. Ponudnikom spletnih portalov na ta način ni treba ročno pregledovati spornih ali pogosto neprimernih vsebin. Razpoznavanje objektov lahko nudi tudi pomoč uporabnikom aplikacij, tako da jih informira, usmerja ali nudi dodatne infor-

macije. Tako lahko nudi pomoč tudi ljudem s posebnimi potrebami. Vse to je mogoče z uporabo nevronskih mrež. Med drugimi področji uporabe prevladuje denimo marketing [9], prav tako je mogoča uporaba v menedžmentu, kjer nudi pomoč pri vodenju poslovanja, kar je predstavljeno v raziskavi [34]. V članku avtorji predstavijo različne načine aplikacij nevronskih mrež v poslovanju in primerjajo učinkovitost tovrstnih aplikacij na področju poslovanja. Nevronske mreže se uporabljajo tudi na področju bančništva in zavarovanja [5], kjer med drugim zaznavajo prevare, lahko pa tudi prepoznavajo kompleksnejše oblike ali vzorce, ki so prisotni na področjih, kot so medicina in bioinformatika. Teoretično je metodo prepoznavanja objektov mogoče aplicirati na mnogih področjih, kjer aplikacija lahko pomaga pri avtomatizaciji pregledovanja ali sortiranja slik. Z avtomatizacijo lahko analiziramo večje količine podatkov, kot so rentgenske slike, zdravstveni izvidi ali spletne dražbe. Hkrati pa so koristne tudi v vsakdanjem življenju, saj lahko razpoznajo podrobnejše razlike tako med objekti in slikami kot tudi med živalmi in rastlinami. Preskok uspešnosti in učinkovitosti nevronskih mrež na področju razpoznavanja objektov je omogočil uporabo v realnih aplikacijah in s tem odpira nove poti različnim aplikacijam na področju klasifikacije.

Na področju računalniškega vida so najbolj uveljavljene konvolucijske nevronske mreže v kombinaciji z različnimi modeli, kot so AlexNet [19], Vgg [25], Resnet [13], Faster-CCN [24], Inception [29]. Vsak izmed pristopov je zanimiv, saj doprinese različne izboljšave.

1.2 Cilji diplomske naloge

V diplomski nalogi je predstavljen razvoj mobilne aplikacije za klasifikacijo objektov na izbrani domeni. Beleženo je spreminjanje rezultatov in uspešnosti klasifikatorja na podlagi različnih parametrov. Cilj je prenos konvolucijske nevronske mreže za razpoznavanje na realni domeni na mobilno napravo in testiranje ter primerjanje uspešnosti na različnih količini podatkov.

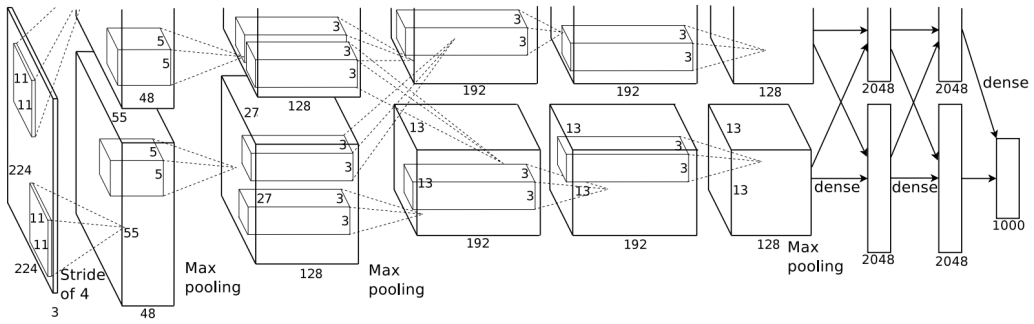
1.3 Pregled področja

Področje razpoznave objektov je zelo obširno in se zaradi velikega interesa industrije hitro razvija. Obstajajo različni pristopi, ki so bili uporabljeni tudi preden so se uveljavile konvolucijske nevronske mreže. Eden izmed predhodnikov je vreča besed (angl. *bag-of-words*) [3], podoben pristop se uporablja za analizo besedil. V primeru slik se razpoznavanje začne z iskanjem regij na podlagi lastnosti in elementov posameznega segmenta slike. Sledi razpoznavanje gradnikov posameznih regij in generiranje opisa, ki najbolj ustreza posamezni regiji. Nato so razpoznane regije in njihovi opise združeni v slovar besed. Na podlagi ustvarjenega slovarja se nato ustvari opis slike. Pristop se je izkazal za uporabnega na različnih področjih, z nekaj omejitvami na podlagi velikosti slovarja, kot je predstavljeno v članku [35].

Za razpoznavanje objektov na sliki se uporablja tudi HOG, metoda histograma orientiranih gradientov, ki jo uporabljamo še danes. V delu [10] predstavijo uporabo HOG v kombinaciji z metodo podpornih vektorjev (angl. *support vector machine*) za detekcijo ljudi. Metoda na podlagi spremembe gradientov razbere izgled in oblike objektov, kar je mogoče na podlagi distribucije gradientov in usmerjenosti robov. Deluje s povzemanjem lastnosti skupka slikovnih točk, katerih vrednosti normalizira. S postopkom normalizacije razpoznavalnik postane bolj odporen na spremembe v variacijah svetlosti in senc. Robusten je v primeru spremembe orientacije in globine detektiranih objektov, vendar se slabo izkaže ob rotaciji le-teh.

Na področju razpoznavanja objektov na slikah že nekaj let poteka tekmovanje ILSVRC [14], katerega rezultati so javno dostopni na njihovi spletni strani [14]. Leta 2012 je nastal AlexNet [19] in s tem je popularnost konvolucijskih nevronskih mrež narasla. Delo [19] je mnogokrat citirano v znanstvenih člankih in velja za eno izmed najbolj vplivnih na svojem področju. S strukturo, ki je prikazana na Sliki 1.1, so leta 2012 zmagali na izzivu ILSVRC. Dosegli so 15,4 % napake, kar je presenetljivo, če upoštevamo, da je naslednji tekmec dosegel zgolj 26,2 % napake. S tem so konvolucijske nevronske mreže postale temelj večine nadaljnjih poskusov. Predor konvolucijskih nevronskih

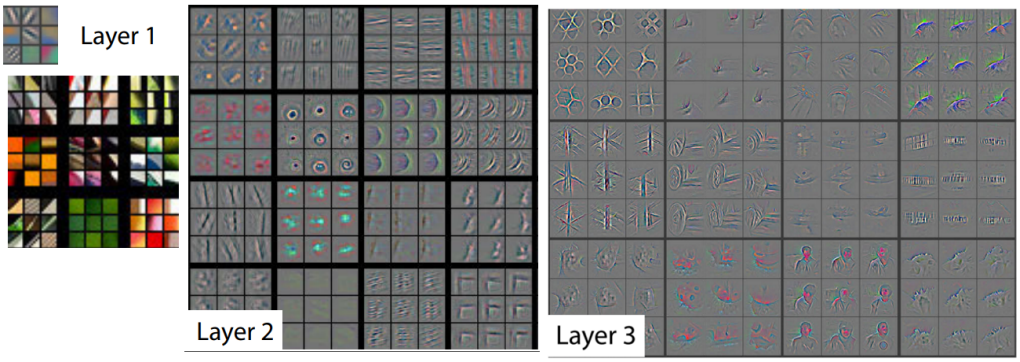
mrež je omogočilo procesiranje na grafičnih karticah, kar je znatno pohitrilo proces učenja.



Slika 1.1: Vizualizacije nevronske mreže AlexNet. Slika povzeta po [19].

Zaradi uspešnosti AlexNeta je v naslednjih letih uporaba CNN poskočila. Naslednja pomembna raziskava je nevronska mreža ZFNet [38]. Pristop s CNN so izboljšali in na tekmovanju dosegli 11,2 % napake ter s tem v letu 2013 zasedli prvo mesto. Mreža je bila priučena (angl. *finetune*) različica AlexNeta, vendar sta avtorja v procesu razvila nekaj ključnih načinov izboljšave. Članek se poglobi v omejeno znanje in razumevanje nevronskih mrež. Glavni prispevki članka so vpogled v podrobnosti spremenjenega AlexNeta in zanimiv prikaz lastnosti (angl. *feature maps*), ki so prikazane na Sliki 1.2. V letu 2014 je sledila konvolucijska nevronska mreža, imenovana VGGNet [25], ki je še danes ena izmed zelo popularnih, zahvaljujoč preprostosti in učinkovitosti. Preizkusili so vpliv globine konvolucijskih nevronskih mrež na uspešnost v klasifikaciji. Bistvo pristopa je bila zamenjava velikega jedra z zlaganjem več manjših konvolucij, pri čemer se uporablja samo konvolucije 3 x 3 v kombinaciji s 3 x 3 združevanjem največjih vrednosti. Izkazalo se je, da lahko s tem pristopom dosežejo dobre rezultate že pri globini 16–19 slojev. Izkazali so se na izzivu ImageNet [14] in v letu 2014 dosegli prvo in drugo mesto v lokalizaciji in razpoznavanju objektov.

Sledila je ResNet [13], ki je v letu 2015 dosegla 3,6 % napake. Težavnost



Slika 1.2: Vizualizacije slojev ZF Net. Slika povzeta po [38].

treniranja globokih nevronskih mrež je spodbudila avtorje, da so predstavili uporabo rezidualnih nevronskih mrež kot olajšanje učenja bistveno globljih mrež kot do sedaj. Na podatkovni množici ImageNet [14] so evalvirali rezidualne mreže z globino do 152 slojev, kar je osemkrat globlje kot VGG [25]. V letu 2015 so s tem pristopom zasedli prvo mesto na tekmovanju ILSVRC in dosegli 3,57 % napake na 1000 kategorijah ImageNet, v katerih najdemo najrazličnejše slike, na primer: trajekti, psi, mačke, avtomobili. Njihov pristop se je kasneje uporabil tudi na področju govora, predstavljen pa je v delu [37].

Leta 2016 so FastRCNN [11] nadgradili s spremembo metode za priporočanje regij, ki je bila uporabljena v MultiBox [28], in z modulom Fast-RCNN Roi Pooling, ki klasificira vsako predlagano regijo in izboljša predlagane lokacije. Pristop so poimenovali Faster R-CNN [24], v njem pa so predstavili mrežo za priporočanje regij (angl. *region proposal network*), ki podane značilke deli z mrežo razpoznavanja in s tem omogoča detekcijo regij skoraj brez dodatnih stroškov procesiranja (angl. *cost-free*). V delu so združili RPN in Fast R-CNN v isto mrežo s skupnimi značilkami z uporabo pristopa nevronskih mrež z mehanizmom pozornosti (angl. *attention mechanisms*), kjer modul RPN služi za usmerjanje pogleda združene nevronske mreže. Z uporabo grafične kartice omogoča detekcijo 5 slik na sekundo in uspešno dosega vrhunske rezultate (angl. *state-of-the-art*) na podatkovni

zbirki PASCAL VOC [21] 2007, 2012 s samo 300 predlogi na sliko.

1.4 Prispevki

Glavni prispevek diplomske naloge je razvoj mobilne aplikacije za razpoznavanje objektov na sliki. Z njo lahko razpoznamo različne kategorije na podlagi zadostnega števila učnih primerov. Uporabimo pristop prenosa znanja na novo domeno, kar omogoča hitrejše učenje in uporabne rezultate z majhnim številom učnih slik. Pristop temelji na uporabi konvolucijskih nevronskih mrež in uporabe zaporednih modulov Inception [29], ki dosega zelo dobre rezultate tudi na zbirkah v velikosti do 1000 kategorij [29]. Za prenos nevronske mreže na mobilni telefon je potrebna dodatna optimizacija. Diplomsko nalogo smo podkrepili z analizo in primerjanjem uspešnosti razpoznavanja na realnem problemu aplikacije razpoznavanja specifične domene v muzejskih zbirkah. Preverjamo vpliv števila učnih razredov, količine podatkov in števila iteracij na sposobnosti modela Inception v3 [29] na uspešno klasifikacijo na izbrani domeni.

1.5 Struktura dela

Diplomska naloga je sestavljena iz petih poglavij. V Poglavju 1.1 predstavimo zgodovino in napredke na področju prepoznavanja objektov ter njihove prispevke. V Poglavju 2 opišemo uporabljene metode. V prvem delu poglavja predstavimo osnovne gradnike nevronskih mrež in njihov matematični zapis. Opišemo konvolucijske nevronske mreže, ki jih uporabimo za kasifikacijo objektov na sliki. Prav tako razložimo pristop Inception v3 [29], na implementaciji katerega temelji diplomska naloga. Predstavimo tudi okolje, ki smo ga uporabili za prenos znanja na izbrano domeno (učenje). Omenimo tudi orodja, ki smo jih uporabili za izboljšanje in optimizacijo nevronske mreže ter za prenos le-te v mobilno aplikacijo. V Poglavju 3 opisujemo potrebno predhodno pripravo učnih podatkov in podrobnosti procesa učenja

zadnjega sloja nevronske mreže. V prvem delu Poglavja 4 predstavimo izbrano zbirko. Nadaljujemo z vpogledom v vplive posameznih parametrov na uspešnost razpoznavalnika. V drugem delu opišemo rezultate poskusa izboljšave (angl. *fine tune*) nevronske mreže s pomočjo umetnega povečanja učne množice. V Poglavju 5 povzamemo rezultate in predstavimo možnosti za nadaljnje delo.

Poglavje 2

Metode

V tem poglavju je opisan koncept konvolucijskih nevronskih mrež, ki so zasnovane na umetnih nevronskih mrežah in simulirajo delovanje nevronov. Gradniki nevronskih mrež delujejo na podlagi vhodnih podatkov in v kombinaciji z različnimi aktivacijskimi funkcijami aktivirajo povezave med umetnimi nevroni, ki na podlagi teh aktivirajo druge vse do zadnjega sloja nevronske mreže. V poglavju tudi opišemo model Inception v3 [29], predstavimo njegovo sestavo in postopek prenosa znanja, na katerem temelji diplomska naloga. Posvetimo se zadnjim slojem klasifikatorja, katere tudi nadomestimo oziroma ponovno učimo in nam omogočajo prenos znanja na izbrano domeno. Prav tako opišemo okolje, ki smo ga uporabili za razvoj in primerjavo rezultatov uspešnosti. Opisane so tudi mobilna aplikacija in optimizacije klasifikatorja za integracijo in prilagoditev za mobilne naprave.

2.1 Zgradba nevronskih mrež

Konvolucijske nevronske mreže so zasnovane na umetnih nevronskih mrežah, ki so sestavljene iz skupkov nevronov. Idejni razvoj nevronskih mrež je imitacija delovanja možganov, ki ob prejemanju nevronskih dražljajev ter ob specifičnih pogojih pošljejo impulze drugim nevronom. Matematični zapis

delovanja nevrona je povzet po [8] in je zapisan kot

$$y = f\left(\sum w_i x_i + b\right), \tag{2.1}$$

kjer w_i predstavlja vrednost uteži, x_i signal in b prag, katerih seštevek na podlagi aktivacijske funkcije preslika vhodne vrednosti. Z uporabo klasične aktivacijske funkcije je izhod binaren, medtem ko pri funkciji ReLu [8], ki jo zapišemo

$$f(x) = \max(0, x), \tag{2.2}$$

lahko dobimo boljše rezultate. Funkcija je preprosta, aktivirana je na podlagi vhodnih parametrov, ki presegajo vrednost 0. Prednosti uporabe funkcije so: hitra konvergenca pri stohastičnega spustu po gradientu (angl. *stochastic gradient descent*) v primerjavi z drugimi funkcijami in preprosta implementacija s pragom aktivacije v ničli. Slabost funkcije je njena krhkost v fazi učenja, saj je mogoče, da se ne glede na vhodne parametre ne aktivira (angl. *dead*). Do tega lahko pride v primeru velikih gradientov, ki jih funkcija prejme v fazi učenja, in uteži posodobi tako, da se nevron ne bo nikoli več aktiviral. To pogosto povzroči previsoka hitrost učenja (angl. *learning rate*).

Nevronske mreže učimo z minimizacijo kriterijske funkcije. V implemetaciji je uporabljena križna entropija (angl. *cross entropy*), povzeta po [8]. Učimo jih z vzvratno propagacijo napake, ki s spreminjanjem uteži preizkuša, kakšne vrednosti najbolje opišejo posamezni razred. To je mogoče zaradi kriterijske funkcije, ki na podlagi rezultatov izračuna napako. Zapišemo jo

$$L_i = -f_{y_i} + log \sum_j e^{f_j}, \tag{2.3}$$

kjer f_j zastopa vrednost j–tega elementa v vektorju razredov f. Medtem f_{y_i} predstavlja odmik (angl. *bias*). Napaka celotnega učenja je povprečje L_i, skozi vse učne primere združeno v regularizacijski člen $R(W)$, ki ga imenujemo vektor z v enačbi (2.4). Aktivacijska funkcija, v kateri ga uporabljamo, je v zadnjem sloju nevronske mreže in se imenuje softmax. Povzeta po [8], kjer predstavijo še druge aktivacijske funkcije. Funkcija `softmax` je definirana

kot

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}, \tag{2.4}$$

njeno izhodišče je skalar $\mathbf{z} = [z_1, \ldots, z_j, \ldots, z_N]$, kjer N predstavlja število vseh vhodnih signalov. Navedena funkcija stisne k–dimenzionalni vektor števil v spodnjem delu enačbe. Spremenljivka k predstavlja število vhodnih vrednosti, z_j zastopa arbitrarno realno vrednost izhoda posameznega nevrona. Vektor z realnih števil je v razponu med vrednostmi $[0, 1]$, vsota katerih je 1. Z verjetnostjo predstavimo izhodno vrednost funkcije softmax, ki je distribucija verjetnosti k različnih možnosti. Uporabljamo jo lahko za različne večrazredne klasifikacijske probleme, kjer želimo pridobiti verjetnost pripadnosti posameznim razredom.

2.2 Sloji nevronskih mrež

Nevroni v možganih so med seboj kompleksno povezani, kar lahko simuliramo z različnimi pristopi. Nekateri izmed njih so rekurzivne nevronske mreže [6], konvolucijske nevronske mreže [26] in umetne nevronske mreže [1]. Simuliramo jih z združevanjem večjega števila nevronov ter tako ustvarimo sloj nevronske mreže. Sloji so med seboj povezani v aciklični graf; izhodne vrednosti nekaterih nevronov so vhodne vrednosti deleža drugih nevronov. Cikli niso dovoljeni, saj bi ustvarili neskončno zanko v usmerjeni nevronski mreži (angl. *feed forward network*). Pri običajnih nevronskih mrežah se najpogosteje uporabljajo povsem povezani sloji (angl. *full-connected layers*) v katerih so nevroni med sosednjimi sloji vsi povezani, medtem ko si v sloju samem ne delijo povezav. V možganih se posamezne povezave med nevroni nenehno krepijo ali slabijo. V umetnih nevronskih mrežah to simuliramo med učenjem s posodabljanjem uteži, kar spreminja povezave med nevroni z uporabo metode, ki se imenuje vzvratna propagacija (angl. *back-propagation*) [4]. Metoda omogoča učenje nevronske mreže z vstavljanjem označenih podatkov. Na podlagi dobljenih rezultatov primerno posodobimo uteži. To storimo s premikanjem nazaj po nevronski mreži.

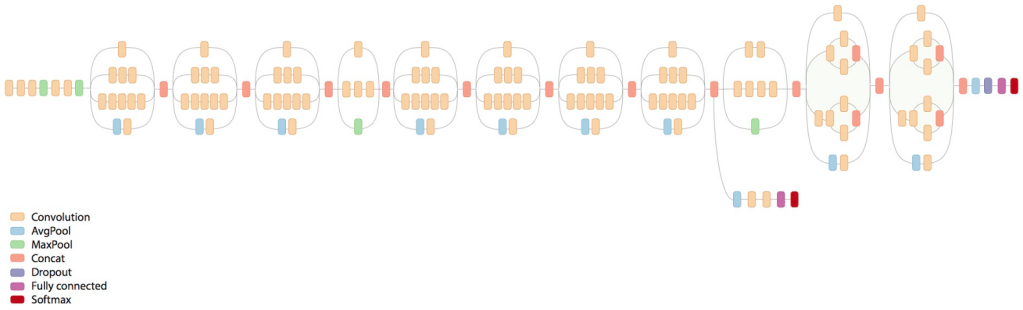
2.3 Konvolucijske nevronske mreže

Konvolucijske nevronske mreže so zelo podobne umetnim nevronskim mrežam. Sestavljene so iz nevronov, ki se učijo s pomočjo uteži in odmika (angl. *bias*). Na eni strani vstavimo slikovne točke slike in na izhodu pridobimo pripadnost posameznemu razredu. Prednost konvolucijskih nevronskih mrež pred umetnimi je predpostavka, da je na vhodu slika. Zaradi tega lahko v arhitekturo implementiramo nekaj ključnih lastnosti. Na podlagi teh je lahko usmerjena funkcija bolj učinkovita in zmanjša število parametrov v mreži. Pri klasifikaciji objektov na sliki so se najbolj izkazale konvolucijske nevronske mreže, ki jih uporabljamo tudi pri procesiranju videa [17], govora [37] in besedila [7]. Idejna zasnova konvolucijskih nevronskih mrež za klasifikacijo objektov na sliki je vidni korteks, kot lahko razberemo iz članka [23]. Posamezni nevroni se odzovejo na stimulacijo samo znotraj določenega področja vidnega polja. Sprejemna polja različnih nevronov se delno prekrivajo in tako pokrivajo celotno vidno polje. Razlog za uspešnost nevronskih mrež na tem področju je tudi količina dela s pred procesiranjem, katerega je v primerjavi z drugimi klasifikacijskimi algoritmi manj. Predhodni algoritmi za predprocesiranje slik so bili ročno zgrajeni, med tem ko se nevronska mreža teh lastnosti nauči sama.

2.4 GoogLeNet Inception v3

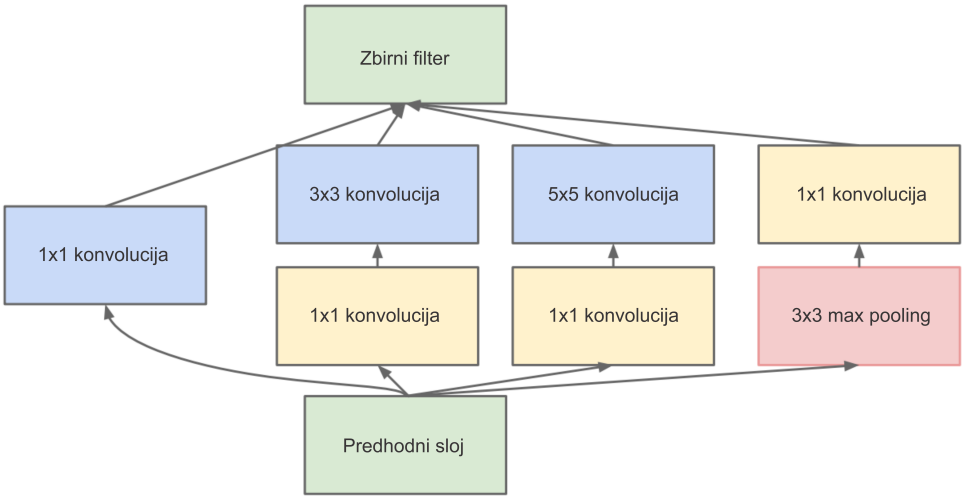
V tem poglavju na kratko opišemo konvolucijsko nevronsko mrežo GoogLeNet Inception v3 [29], saj smo jo uporabili pri našem pristopu in predstavlja osrednji element našega razpoznavalnika objektov na slikah. V letu 2014 je Google uporabil nevronsko mrežo z 22 sloji in zmagal s 6,7 % napake. Preizkusili so nov pristop in predstavili uporabo modula, ki se imenuje „Inception module“. Avtorji članka poudarjajo, da model poveča porabo spomina, procesorskega časa in je bolj naklonjen k pretiranem prileganju učnim podatkom. Slika 2.1 prikazuje pregled strukture GoogLeNet Inception v3.

Model je sestavljen iz več sekvenčnih členov. Posamezni člen se imenuje



Slika 2.1: Pregled strukture GoogLeNet Inception v3.

Inception module in je sestavljen iz več elementov. Na Sliki 2.2 so podrobneje



Slika 2.2: Modul Inception iz članka [27].

prikazani gradniki modula. Spodnji in zgornji okvir predstavljata vhodne in izhodne signale. Modul je sestavljen iz več manjših komponent ali konvolucij, kot so 1 x 1, 3 x 3 in 5 x 5 skupaj s 3x3 *max pooling*. Razlogov za tako strukturo je več. Manjši konvolucijski moduli lahko zaznajo majhne vzorce, medtem ko lahko modul 5 x 5 pokriva večji del vidnega polja. Vsakemu modulu sledi modul ReLu, ki skrbi za nelinearnost mreže, kar pomeni,

da ohranjamo računsko kompleksnost. Zahvaljujoč tem modulom Inception vsebuje 100 slojev in ohranja dvanajstkrat manj parametrov kot AlexNet [19].

2.5 Ogrodje za razvoj aplikacije

Izdelati smo želeli aplikacijo za mobilne naprave, ki bo omogočala razpoznavanje objektov s konvolucijsko nevronsko mrežo. Za generiranje nevronske mreže smo potrebovali orodje, ki to omogoča. Na voljo jih je veliko; nekateri izmed njih so Matlab [20], Theano [32], Weka [36], Torch [33], Infer.NET [16], Keras [18] in Tensorflow [30]. Vsako izmed njih ima svoje prednosti in slabosti, za našo aplikacijo smo se odločili za uporabo okolja Tensorflow, saj omogoča razvoj nevronske mreže in preprosto integracijo le-te v mobilno aplikacijo. Ogrodje Tensorflow je odprtokodna knjižnica za numerične operacije, specializirana za področje strojnega učenja. Fleksibilna infrastruktura omogoča procesiranje na enem ali več grafičnih ali računalniških procesorjih. Prednost je tudi uporaba tako na osebnih računalnikih kot tudi na strežnikih in mobilnih napravah. Knjižnica je bila zasnovana za raziskovalce in razvijalce, ki so sodelovali pri ekipi Google Brain Team [30] znotraj raziskovalne organizacije Google's Machine Intelligence. Le-ta je zadolžena za raziskave na področju globokega učenja in globokih nevronskih mrež. Skozi čas je sistem postal dovolj razsežen in uporaben za širšo rabo na drugih domenah. Uporabljamo jo lahko na različnih operacijskih sistemih Ubuntu, Mac OS X in Windows, ki so uradno podprti s strani razvijalcev. Knjižnica je razvita v C++ in uporabljena v Pythonu, a v okrnjenih različicah obstaja tudi za druge programske jezike, kot so C, Java in Go.

Pri procesiranju na različnih napravah in sistemih pride do težav v kompatibilnosti različnih sistemov, zato smo uporabili okolje Docker na operacijskem sistemu Ubuntu. Docker omogoča uporabo razvojnega okolja Linux v neodvisnosti od preostalega računalniškega sistema. Z uporabo vnaprej

definiranega okolja Tensorflow na osnovi Dockerja, ki nam zagotavlja vse potrebne odvisnosti, nam ni treba skrbeti za kompatibilnost in različice operacijskega sistema, gonilnikov in programov. S spreminjanjem nastavitev in konfiguracije lahko uporabimo tudi procesorsko enoto grafične kartice in s tem še dodatno pospešimo opravljanje velikega števila operacij.

Za urejanje, spreminjanje in namestitev mobilne aplikacije Android uporabljamo okolje Android Studio [2]. Namenjeno je kvalitetnemu in hitremu razvoju aplikaciji, omogoča tudi odpravljanje napak, testiranje in emulacijo operacijskega sistema Android. Pri razvijanju mobilne aplikacije moramo biti pozorni na podprte verzije operacijskega sistema Android. Posledično aplikacija ne bo dosegljiva vsem uporabnikom.

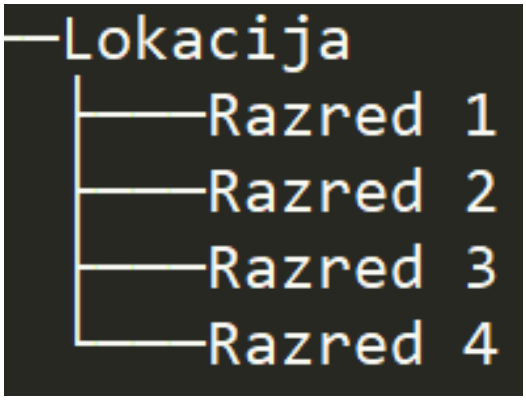
2.6 Skripta za učenje

Skripta [22], ki omogoča ponovno učenje razpoznavalnika Inception v3 [29], objavljena s strani Googla, omogoča določitev nekaj variabilnih parametrov, s katerimi lahko spreminjamo potek procesa učenja. Proces učenja začnemo s:

```
python -m scripts.retrain \
  --bottleneck_dir=bottlenecks \
  --how_many_training_steps=2000 \
  --output_graph=retrained_graph.pb \
  --output_labels=retrained_labels.txt \
  --image_dir=Lokacija
```

Nato skripta začne s procesiranjem. Na slednje vplivamo z parametri. S parametrom `--image_dir=Lokacija` podamo lokacijo razredov. Na Sliki 2.3 je primer drevesne strukture map razredov, katerih mape vsebujejo slike, ki smo jih prilagodimo za učenje nevronske mreže. Mape služijo za poimenovanje posameznih razredov. Pri zagonu učenja nam je na voljo tudi parameter `--how_many_training_steps`, s katerim določimo število iteracij. Parameter

`--output_graph=retrained_graph.pb` določi mesto za shranitev grafa. Parameter `--output_labels=retrained_labels.txt` določa lokacijo, v katero se shranijo imena kategorij v tekstovni obliki. Ko generiramo učne datoteke za posamezne slike, jih ni treba ponovno generirati, saj so za primer ponovnega učenja shranjene v mapi `--bottleneck_dir=bottlenecks`. Ob dodajanju novih razredov se za učenje pripravi le nov razred ali na novo dodani podatki. Na učni proces razpoznavalnika lahko vplivamo tudi z upo-



Slika 2.3: Slika je primer drevesne strukture mape z razredi.

rabo dodatnih parametrov, kot so naključna deformacija, svetlost ali obrezovanje učnih slik. Skripta omogoča distorzije z uporabo zastavic, kot so `random_crop`, `random_scale` in `random_brightness`. Vrednosti so v procentih in v fazi testiranja lahko začnemo z vrednostmi od 5 do 10. Vrednosti so spremenljive za vsako domeno in treba je eksperimentirati, koliko doprinesejo aplikaciji. Obstaja tudi zastavica `flip_left_right`, ki naključno zrcali sliko po horizontali, kar je super, razen v primeru inverzij v aplikaciji. Na primer ni učinkovito zrcaliti učnih podatkov, ko poizkušamo klasificirati pisma, saj s tem izgubijo namembnost (pomen).

2.7 Hiper-parametri

Obstaja še nekaj parametrov, ki jih lahko spremenimo, da izboljšamo rezultat učenja klasifikatorja. Eden izmed njih je hitrost učenja (angl. *learning rate*), ki nadzira stopnjo posodobitev predzadnjega sloja med učenjem. Manjša kot je, dalj časa se bo klasifikator učil, vendar lahko v končni fazi izboljša gotovost rezultatov. Temu pa vedno ni tako; potrebnih je nekaj preizkusov in šele nato je razvidno, ali to pripomore v naši domeni. Parameter velikosti učnih množic določa, koliko učnih slik je preučenih v vsakem koraku. Ker stopnjo učenja določimo za vsak paket posebej, ga je treba zmanjšati, ko povečamo pakete, da dobimo primerljive rezultate. Če opazimo, da se validacijska gotovost (angl. *validation accuracy*) zelo spreminja z vsako iteracijo, lahko to pripišemo naključno izbrani potrditveni množici pri vsaki meritvi. Velikim spremembam se lahko izognemo z uporabo celotne potrditvene množice za vsako meritev gotovosti, vendar s tem podaljšamo čas učenja.

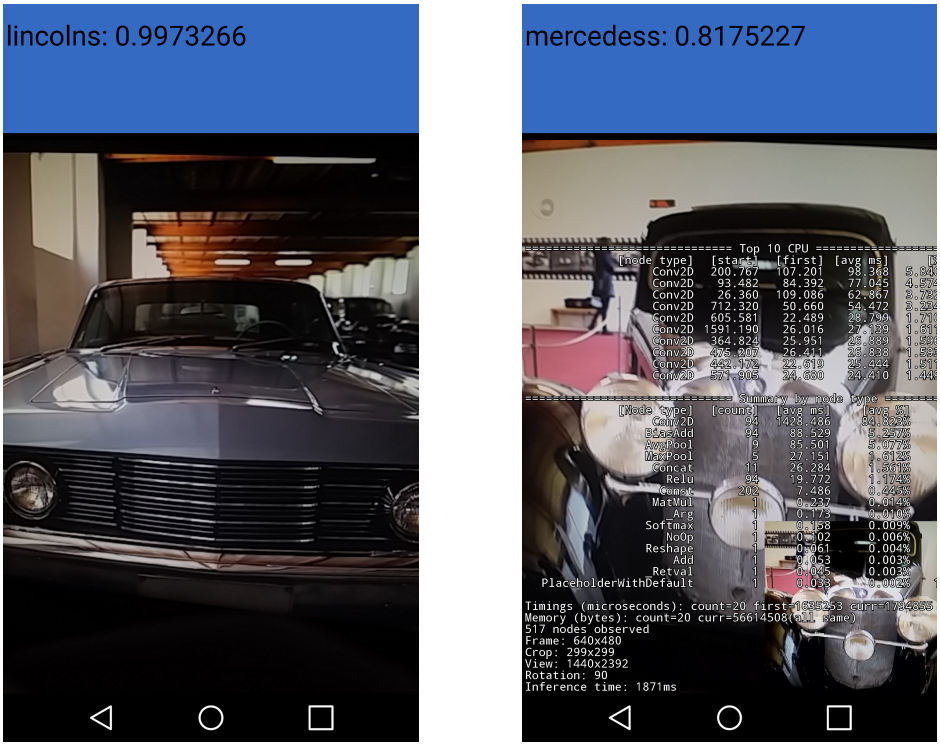
2.8 Prenos na mobilne naprave

Povzetek procesa prenosa grafa na mobilno napravo je prikazan na Sliki 2.4. Proces je sestavljen iz treh delov. V prvem naučimo in optimiziramo graf. Nato ga vključimo v projekt mobilne aplikacije s pomočjo okolja Android Studio [2]. Nazadnje to generirano aplikacijo naložimo na mobilno napravo. Generirani model je za mobilne naprave preveč prostorsko zahteven, saj potrebuje 89 MB prostora. Na mobilnih napravah smo na različne načine omejeni, zato je zaželeno, da vnaprej pripravimo in optimiziramo aplikacijo, kolikor je to mogoče. Že knjižnica Tensorflow je za mobilne naprave manjša. To dosežejo tako, da zagotavljajo le omejene funkcionalnost. Ena izmed izpuščenih je treniranje, saj le redko treniramo model na mobilnih napravah. Druga izmed funkcionalnosti je podpora dekripcije slikovnega formata JPEG, saj je podpora le-te na mobilnih napravah zahtevna in poveča velikost aplikacije. Za procesiranje na telefonu je ne potrebujemo, saj lahko delamo direktno s predpomnilnikom kamere. Zato nam je na voljo skripta

Slika 2.4: Proces prenosa razpoznavalnika na mobilno napravo.

`optimize_for_inference`, del večjega sklopa orodij [31], ki omogočajo optimizacijo nevronske mreže za mobilne naprave. Metoda odstrani vsa vozlišča nevronske mreže, ki se ne uporabljajo pri danih vhodnih in izhodnih vozliščih. Graf kljub optimizaciji še vedno zaseda 84 MB prostora, vendar se vsaka aplikacija ob distribuciji stisne, zato lahko z uporabo `gzip` [12] preverimo, kolikšen bo graf po stiskanju. Velikost se ne spremeni veliko, le za 7 %. Potrebna je dodatna optimizacija. Večino prostora zasedajo uteži, ki jih hranimo v formatu `float`. Graf lahko kvantiziramo, kar pri slikah pomeni manjšanje števila barv. Pri utežeh je podobno; z uporabo orodja kvantizacije dodamo več ponavljanja med vrednostmi uteži. Graf lahko zahvaljujoč temu dodatno stisnemo, s tem zmanjšamo gotovost, ampak le za slab odstotek. Struktura grafa v tem procesu ostaja enaka. Proces začnemo s skripto `quantize_graph` [31] in ji podamo lokacijo grafa, ki ga želimo optimizirati. Graf stisne za 73 %. To smo dosegli brez spreminjanja grafa in prevelikega negativnega vpliva na gotovost. Za preverjanje delovanja uporabimo demo aplikacijo, ki je na voljo skupaj z okoljem Docker [31]. V le-to lahko dodamo graf in oznake razredov, tako da jih premaknemo v mapo `android/assets` demo aplikacije.

Za spreminjanje in pretvorbo aplikacije potrebujemo program Android Studio [2]. Aplikacija je na telefonu preprosta. Na Sliki 2.5a je prikazana pravilna klasifikacija avtomobila Lincoln. Na zajeti sliki se v zgornjem delu



(a) (b)

Slika 2.5: Demo aplikacija na telefonu z operacijskim sistemom Android.

zaslona v modrem območju izpisuje ime razreda in gotovost klasifikacije, ki je priročna informacija pri testiranju razpoznavalnika in preizkušanju mobilne aplikacije. Na Sliki 2.5b so prikazani aktivacija posameznega sloja in nekaj dodatnih tehničnih podatkov, kot sta resolucija in čas, ki ga porabimo za klasifikacijo posamezne slike.

Poglavje 3

Razvoj mobilne aplikacije

To poglavje opisuje različne metode in pristope, ki smo jih uporabili za razvoj razpoznavalnika na domeni muzejske zbirke avtomobilov in za povečanje njegove uspešnosti. Moderni pristopi prepoznavanja objektov imajo milijone parametrov, ki jih trenirajo več tednov. Prenos znanja je bližnjica, ki nam olajša delo, tako da naknadno učimo že naučeno nevronsko mrežo. Mreža Inception v3 [29] je naučena na zbirki kategorij ImageNet [14] in smo jo zgolj doučili prepoznavanja na novih razredih. Opisujemo postopek, v katerem zamenjamo softmax, ponovno pa se uči zgolj predzadnji polno povezani sloj; vsi drugi ostanejo nedotaknjeni. Ni tako učinkovito kot popolno učenje, vendar so že naučene uteži prenosljive na mnogo različnih področij. Navkljub kratkotrajnemu učenju lahko na boljšem prenosnem računalniku celo brez uporabe grafične kartice v zgolj 30 minutah doseže dobre rezultate. Primeri ponovnega učenja in dobre prakse so povzeti iz [15].

3.1 Implementacijske podrobnosti

Uporabljena je nevronska mreža Inception v3 [29], katere arhitektura je prikazana na Sliki 3.1. Ponovno učimo le predzadnji polno povezani sloj arhitekture, postopek pa se imenuje prenos znanja na izbrano domeno. To dosežemo tako, da že naučenih slojev kovolucijske nevronske mreže ne spre-

minjamo, saj služijo za razpoznavanje značilnic in s tem omogočajo uporabo bližnjice. Na podlagi tega lahko učimo samo predzadnji polno povezani sloj klasifikatorja, ki razpozna razrede izbrane domene. Nevronska mreža je implementirana s konvolucijskimi sloji, kateri so prikazani na Sliki 3.1. Vstopni sloj v mrežo je konvolucija 3 x 3, ki sprejme sliko velikosti 299 x 299 slikovnih točk v barvnem prostoru RGB. Prvih nekaj slojev nevronske mreže služi za manjšanje števila potrebnih računskih operacij, zato smo podatke, preden pridejo do zahtevnejših konvolucijskih slojev, zmanjšali za nekajkrat. To je prispevalo k manjši računski kompleksnosti in k uspešnosti nevronske mreže. To je mogoče zaradi ponavljajočih se vzorcev, ki jih primerno združimo v konvolucijske sloje 1 x 1. Vmesni moduli Inception so variacije primera, prikazanega na Sliki 2.2, ki uporabljajo različne velikosti konvolucije za razpoznavanje večjih in manjših vzorcev na slikah. Velikost izhodnega vektorja vsakega sloja se ujema z velikostjo vhodnega vektorja naslednjega sloja. Na koncu nevronska mreža združuje povprečne vrednosti, katere funkcija softmax normalizira v posamezne razrede, ki smo jih ob začetku učenja določili s strukturo map. Vrednosti so med 0 in 1, kar odslikuje gotovost sovpadajočega razreda.

type	**patch size/stride** or remarks	**input size**
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 4	35×35×288
5×Inception	As in figure 5	17×17×768
2×Inception	As in figure 6	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Slika 3.1: Slika prikazuje sloje nevronske mreže Inception v3 iz [29].

3.2 Priprava učnih primerov

Osnovni postopek ponovnega učenja je priprava učnih podatkov in organizacija le-teh v kategorije. Šele nato lahko začnemo s procesiranjem. Na podlagi definiranih parametrov in količine podatkov bo učenje razpoznavalnika z manjšim številom učnih podatkov (300 učnih primerov) potekalo 20 minut, v primeru večjega števila pa tudi do 50 minut. Preden začnemo s procesiranjem podatkov moramo biti pozorni, da jih pravilno organiziramo. To pomeni, da ima vsaka podmapa ime, s katerim opišemo posamezno kategorijo, vsebuje pa lahko le slike, ki spadajo v izbrano kategorijo. V nasprotnem primeru bomo dobili slabe rezultate in nizko gotovost razpoznanih kategorij. S spreminjanjem učnih parametrov in preizkušanjem (angl. *trial and error*) lahko ugotovimo, kaj dobro ali slabo vpliva na gotovost razpoznavalnika na specifični domeni. Eden izmed prvih problemov, s katerim se soočimo, je zbiranje podatkov, na katerih želimo nevronsko mrežo učiti. Za učinkovito učenje je potrebno zbrati več sto slik vsake kategorije, ki jo želimo prepoznati. Več slik kot zberemo, večja je verjetnost, da bo klasifikator uspešen pri klasificiranju razredov na slikah, ki jih ni bilo v učni množici. Treba je tudi zagotoviti, da slike dobro predstavljajo problem, ki ga rešujemo. Če so na primer vse slike objekta v zaprtih prostorih s pustim ozadjem, uporabniki pa bodo aplikacijo uporabljali v naravi, potem rezultati ne bodo zadovoljivi. Pogosto se ne moremo izogniti temu, da se mreža že med učenjem nauči vsega, kar imajo učne slike posameznega razreda skupnega, in kar morda za nas ne bo uporabno. Če na primer en objekt slikamo v modri sobi in drugega v zeleni, se bo model naučil klasificirati slike na podlagi ozadja in ne glede na lastnosti objekta samega. Da se temu izognemo, poskušamo zajeti slike pod različnimi pogoji in v največ možnih različnih situacijah ob različnih časih in z različnimi napravami.

3.3 Določanje razredov

Priporočljivo je tudi razmisliti o kategorijah, ki jih uporabljamo. Morda se splača večje kategorije, ki pokrivajo veliko različnih fizičnih oblik, razdeliti v manjše, ki se vizualno razlikujejo med sabo. Tako lahko na primer namesto kategorije vozila uporabimo kategorije avto, motor, tovornjak. S tem ločimo podobne razrede in iz porazdeljene gotovosti pridobimo večjo gotovost za posamezno kategorijo. Razmisliti je treba tudi, ali rešujemo problem odprtega ali zaprtega sveta. V zaprtem svetu so namreč vsi objekti in razredi, ki jih klasificiramo, znani. To je običajno pri klasificiranju cvetlic, prostorov in avtomobilov, kjer vemo, da bodo uporabniki najverjetneje slikali motive, ki spadajo v to domeno, mi pa moramo določiti, v katero podkategorijo spadajo. Ravno nasprotno je pri potujočem robotu, ki bo skozi kamero na svoji poti po svetu opazil mnogo različnih stvari. V zadnjem primeru bi si želeli, da klasfikator poroča, če so mu objekti, ki jih opazuje, neznani. Težko je zagotoviti, da bomo problem rešili dobro. Ponavadi zadostuje, če zberemo veliko količino slik, ozadja, ki nimajo relevantnih objektov, in te slike nato dodamo v razred 'neznano', v mapo med druge razrede. Priporočljivo je tudi preveriti, ali res vse slike nahajajo v pravih razredih. Podatki, ki jih označijo uporabniki, so pogosto nezanesljivi. Na primer pri sliki marjetice uporaba poimenovanja osebe Marjetica. Če preiščemo vse slike za napačno opredeljene primere, lahko naredimo drastično izboljšamo splošno gotovost.

3.4 Zajem učnih podatkov

Učni podatki so bili zajeti z fotoaparatom Canon D5 Mark II, v resoluciji 1920 x 1088 pikslov in s hitrostjo 25 slik na sekundo (angl. *frames per second*). V povprečju smo zajeli 1 minuto videoposnetka za posamezni razred, skupaj je vseh slik 23000, kar je približno 1500 slik na razred. Vse slike niso bile uporabljene v učnih razredih, nekatere smo zaradi dvoumnega pogleda odstranili. Posnetke smo za imitacijo primerljivih pogojev, kot bodo kasneje v uporabi pri razpoznavalniku, zajeli v zaprtih prostorih brez dodatnega osve-

tljevanja. Praviloma je priporočljivo zajeti slike v pogojih, ki so primerljivi s pogoji v času uporabe. Na različnih ozadjih s tem preprečimo, da bi se razpoznavalnik naučil klasifikacije na podlagi okolice. Z več različnimi napravami tako dodamo nekaj variacije v podatkih, saj vsaka kamera deluje nekoliko drugače. Če so objekti na sončni svetlobi, to storimo v jutranjem času, ko je svetloba mehkejša. Testne podatke smo zajeli z dvema različnima mobilnima napravama. Na Sliki 3.2a je testni primer zajet z mobilno napravo Apple v resoluciji 3264 x 2448 pikslov, na Sliki 3.2b pa je zajet z mobilno napravo LG velikosti 1920 x 1080 pikslov. Testne slike so tudi različno oddaljene in rotirane, kot je prikazano na Sliki 3.3.

(a) (b)

Slika 3.2: Testna primera, zajeta z mobilnima napravama.

Slika 3.3: Slika prikazuje razred Fiat.

3.5 Pred-procesiranje učnih primerov

Z izbiro določenega razreda, na katerem želimo klasifikator učiti, smo učne slike ločili v posamezne mape na podlagi razredov, ki jih predstavljajo. Imena slik niso pomembna, medtem ko imena map, v katerih se nahajajo, so, saj le-te postanejo oznake razredov. V prvi fazi skripta analizira slike na disku in za vsako izmed njih izračuna ozko grlo (angl. *bottleneck*) vrednosti. Ozko grlo je neformalen izraz za predzadnji sloj tik pred slojem, ki klasificira. To pomeni, da vsaka slika, ki vstopi v nevronsko mrežo, potuje do predzadnjega sloja, ki nato vrne vrednosti. Sloj je naučen, da na podlagi predhodnih vrednosti nevronov vrne nove vrednosti, ki zadoščajo za razlikovanje med posameznimi želenimi razredi. To pomeni, da mora biti povzetek kompakten in sestavljen le iz ključnih informacij za učinkovite odločitve na majhni množici značilk. Razlog za uspešnost z novimi razredi je, da so značilke, ki so potrebne za klasificiranje, med 1000 razredi ImageNet pogosto uporabne tudi pri razlikovanju med novimi objekti. Ker vsako sliko med učenjem uporabimo večkrat in bi računanje vrednosti ozkega grla vzelo veliko časa, nižjih slojev mreže pa ne spreminjamo, se izognemo vnovičnemu računanju, tako da njihove izhodne vrednosti zapišemo na disk. Posledično jih ob ponovnem zagonu skripte ne bo treba ponovno računati. Ko so ozka grla ustvarjena, se prične učenje predzadnjega sloja nevronske mreže.

3.6 Proces učenja

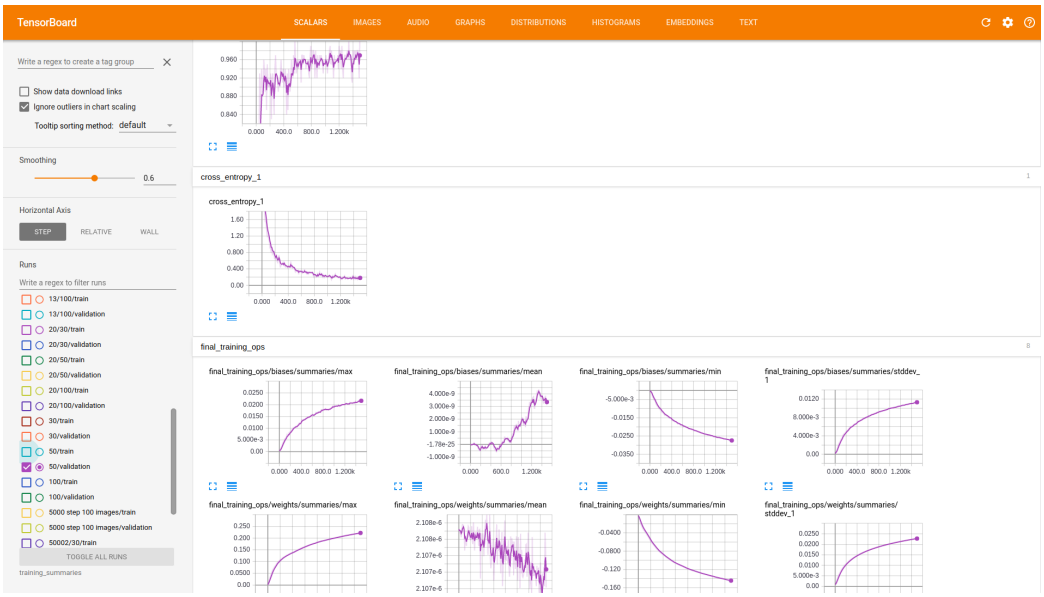
Proces učenja poteka tako, da v vsaki iteraciji vzamemo naključne učne primere, na podlagi katerih se sloj uči klasificirati razrede. Učenje poteka tako, da v vsakem koraku skripta izpiše rezultate učenja. To so gotovost (angl. *accuracy*), točnost vrednotenja in križna entropija (angl. *cross entropy*). Gotovost pove, kolikšen procent primerov je bil v trenutni učni množici (angl. *batch*) klasificiran pravilno. Točnost vrednotenja je gotovost na validacijski množici, ki nikoli ni del učne množice. Učne slike so bile uporabljene za učenje klasifikatorja, na lastnosti katerih se lahko preveč prilagodi; najpo-

gosteje je to šum v učnih podatkih. Točnost vrednotenja pa je množica, ki ostane celoten učni proces ločena od učnih podatkov in jo uporabljamo le kot nevtralen smernik, ki nam pove, ali poteka proces učenja v pravo smer. Ta postopek preverjanje je potreben, saj bolj pravilno in realno preverjanja stanje naučenega, ker testira znanje na podatkih, ki niso bili v učni množici. V primeru, ko je gotovost visoka, točnost vrednotenja pa ostaja nizka, se mreža prekomerno prilagodi na lastnosti v učni množici (šum), to pa negativno doprinese k generalizaciji. Križna entropija (2.3) je kriterijska funkcija, ki nam omogoča vpogled v napredek učnega procesa. Cilj je imeti najmanj napačno opredeljenih slik, saj so iz tega razvidne spremembe in napredki v učenju nevronske mreže, če zanemarimo občasen šum. V vsakem koraku skripta izbere 10 naključnih slik iz učne množice in vzame njihove vrednosti iz vnaprej generiranih datotek, ki jih nato posreduje predzadnjemu sloju, ta pa ji na podlagi le-teh vrne verjetnost pripadanja slike posameznemu razredu. V tem procesu pridobljene razrede primerja s pravimi razredi in z rezultatom primerno posodobi uteži predzadnjega sloja z metodo vzvratnega propagiranja napake. Če so parametri pravilno nastavljeni, funkcija napake globalno pada s številom iteracij. Ko so vse iteracije zaključene, proces preveri uspešnost naučene mreže s testiranjem na podatkih, ki so bili ločeni od učne in testne množice. Ta rezultat je najboljši približek uspešnosti nevronske mreže v klasificiranju danih razredov. Dobljeni rezultati se ob vnovičnem procesiranju med seboj razlikujejo; spremembe se pojavijo zaradi procesa naključnega izbiranja učne in testne množice. Nekaj slik je bilo že na začetku procesa ločenih od učne množice, zato da jih lahko razpoznavalnik ob zaključku klasificira, pridobljeni rezultati so bolj primerljivi z uporabo v realnem svetu.

3.7 Testiranje in vizualizacija

Razvojno okolje Tensorflow omogoča spremljanje kvalitete procesa učenja predzadnjega sloja nevronske mreže s pomočjo orodja TensorBoard, katerega izgled je prikazan na Sliki 3.4. Le-to omogoča vizualizacijo različnih

informacij o procesu, kot so maksimalna, minimalna, povprečna in standardna deviacija uteži ter spremljanje spreminjanja križne entropije in gotovosti skozi iteracije. Vizualizacije nam nudijo tudi vpogled in razumevanje ter oporno točko pri primerjanju rezultatov testiranja. Če želimo preveriti uspešnost naučenega, moramo klasifikator testirati na primerih, ki jih ni bilo v učni množici. To učinkovitost merimo z validacijsko gotovostjo. Če je leta nizka navkljub visoki učni gotovosti, se je izbrani klasifikator prekomerno prilagodil podatkom (angl. *overfit*). Nevronska mreža klasificira lastnosti učnih slik, ki ne pripomorejo pri klasifikaciji bolj generalnih slik.



Slika 3.4: Primer vizualizacije z orodjem TensorBoard.

Poglavje 4

Eksperimenti

4.1 Zbirka slik

Za testiranje smo si izbrali zbirko avtomobilov v Tehniškem muzeju Slovenije v Bistri. V zbirki je 15 vozil, ki so prikazana na Sliki 4.1. Večina vozil je redkih primerkov prestižnih limuzin z visoko tehnično in zgodovinsko vrednostjo. Zbirka ni preprosta za razpoznavanje, saj si je več razredov med seboj podobnih, kot na primer:

- Cadillac in Lincoln,
- ZIS, Mercedes, Horch in Packard.

Vse učne razrede smo ločili v dva najbolj pogosta pogleda avtomobila. Na Sliki 4.2a je prikazan sprednji del razreda Mercedes, na Sliki 4.2b pa je prikazan zadnji del; s tem smo dobili v vsakem razredu med 300 in 600 slik.

4.2 Metode evalvacije

V učnem procesu razdelimo podatke na učno, validacijsko in testno množico. V učni množici je večina podatkov (približno 80 %), preostanek razdelimo med validacijsko in testno množico. Vsaka izmed njiju vsebuje okoli 10 % celotne množice. Učno množico uporabimo za učenje klasifikatorja na

Slika 4.1: Zbirka vozil.

označenih podatkih in na podlagi dobljenih rezultatov temu primerno posodobimo uteži v predzadnjem sloju razpoznavalnika. Validacijska množica služi za sprotno preverjanje uspešnosti klasifikatorja med samim procesom učenja, medtem ko testno uporabimo na koncu in z njo preverimo uspešnost klasifikatorja na še nikoli videnih podatkih. Če imamo na voljo dovolj učnih podatkov, spreminjanje parametrov količine podatkov v posamezni množici nima močnega vpliva na učenje klasifikatorja. Pred začetkom učenja skripta pripravi vrednosti ozkega grla. Pripravi 1000 slik v dveh minutah in treh se-

(a)

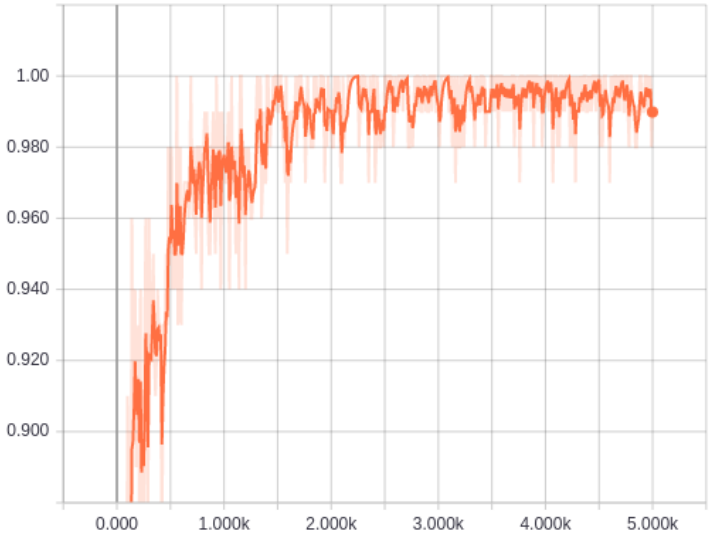
(b)

Slika 4.2: Sprednji in zadnji pogledi avtomobila Mercedes.

kundah na 4–jedrnem prenosniku s sposobnostjo sočasne obdelave dveh niti ukazov na procesorju. Učenje predzadnjega sloja klasifikatorja s 1000 iteracijami, 5 razredi in 200 učnimi primeri na razred skupaj z pripravo podatkov traja 2 minuti in 31 sekund. Priprava 24700 učnih podatkov in učenje klasifikatorja na grafični kartici skozi 1000 iteracij traja 32 minut in 31 sekund. V primeru ponovnega učenja na istih podatkih brez priprave podatkov v obeh primerih zaključimo v minuti in pol. Na računalniku z uporabo procesorja lahko v sekundi razpoznamo 5 različnih slik, z uporabo grafične kartice pa 11 slik. Medtem na mobilnih napravah ne dosegamo takšne hitrosti. Na testni mobilni napravi srednjega cenovnega ranga smo dosegli klasifikacijo ene slike v 1760 ms.

4.3 Izbira števila iteracij v učenju

Ker smo ločili poglede in ima naša domena sedaj 27 razredov, ki so predstavljeni z modelom avtomobila ter z njegovim sprednjim in zadnjim pogledom, smo začeli z vključevanjem vseh v učni proces in s primerjavo potrebnega števila iteracij za doseganje želene gotovosti. Na Sliki 4.3 je prikazan proces učenja klasifikatorja na 27 razredih skozi 5000 iteracij. V našem pri-



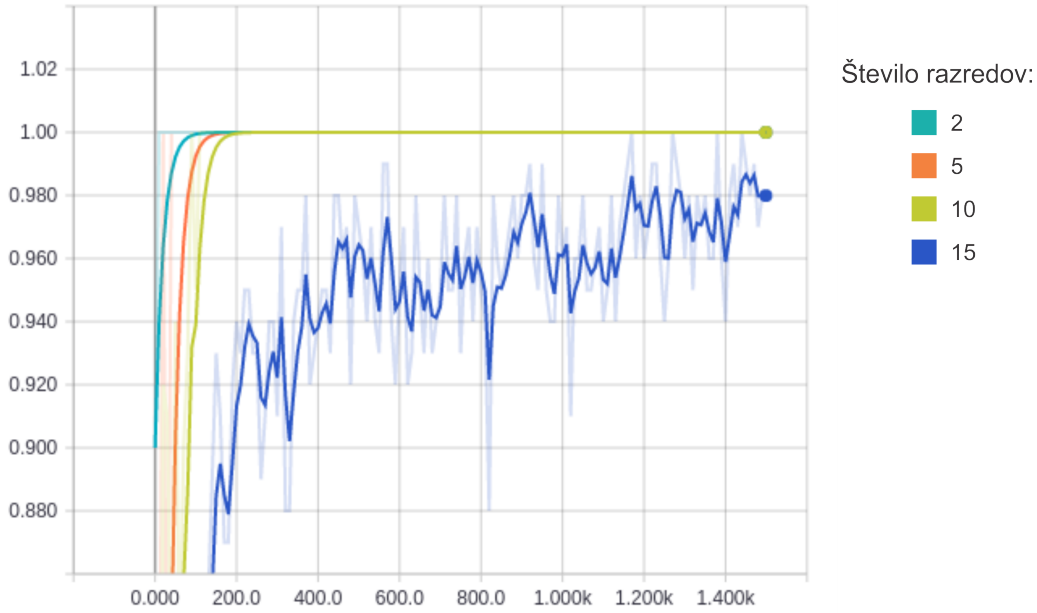
Slika 4.3: Spreminjanje gotovosti skozi 5000 iteracij.

meru klasifikator šele po 500 iteracijah preseže 96 % gotovost. Za naslednjo večjo spremembo, se pravi do 98 %, pa potrebuje še naslednjih 1000 iteracij. Rezultati se ves čas izboljšujejo, vendar se tudi preveč prilagajajo na učne primere. Če si tega želimo, je to optimalno število iteracij. Ker v našem primeru potrebujemo splošen klasifikator, ki dobro pokriva več razredov, in smo zadovoljni z nižjo gotovostjo, vendar si želimo boljšo generalizacijo, se bomo v preostanku posvetili učenju z nižjim številom iteracij. V nadaljnjih primerih je prikazano spreminjanje gotovosti čez 1500 iteracij. Na to število smo se omejili zaradi boljše preglednosti in ker v večini primerov klasifikatorji v kasnejših iteracijah tudi konvergirajo.

4.4 Vpliv števila razredov

Za začetek testiranja učnega procesa smo izbrali za posamezen razred samo 30 učnih primerov. Uporabili smo specifično mrežo, ki je sposobna ločevati med mnogimi kategorijami. Če bi uporabili klasifikator brez predhodnega znanja, bi potrebovali veliko več učnih podatkov. Najprej smo primerjali,

v kakšni relaciji je število razredov s številom učnih podatkov. Na Sliki 4.4



Slika 4.4: Spreminjanje gotovosti pri različnem številu razredov s po 30 učnimi slikami.

najbolj leva (turkizna) črta prikazuje spreminjanje z dvema razredoma. Naslednja (oranžna) prikazuje 5, rumenozelena pa 10 razredov. Pri prvih treh opazimo prekomerno prilagoditev na dane podatke, saj v učnih primerih ni dovoljšne variacije. Le modra črta s 15 razredi ima nekoliko drugačno učno sled z variacijo v uspešnosti klasifikatorja.

Iz Tabele 4.1 vidimo uspešnost klasifikatorja z različnim številom razredov in njegovo uspešnost klasificiranja testnih slik. V primeru razpoznavanja manjšega števila razredov priporočamo uporabo dodatnega razreda. Če razpoznavamo med bananami in jabolki in uspešnost klasifikatorja testiramo z razpoznavanjem banan in jabolk, so rezultati dobri, vendar moramo upoštevati da bodo nekateri uporabniki preizkušali klasifikator tudi kje drugje; nadvse zabavno bo, ko bomo mačko na mizi označili za jabolko. Obstaja tudi večja verjetnost, da bo klasifikator v primeru dvorazredne klasifikacije brez dodatnega razreda napačno klasifikaciral sliko že v primeru majhnega

Št. razredov / pripadnost	`Pravilno`	`Nepravilno`
2	157	11
5	316	41
10	700	98
15	739	143

Tabela 4.1: Primerjava uspešnosti klasifikatorja s 30 učnimi slikami. V prvem stolpcu je število učnih razredov, v drugem stolpcu je število pravilno opredeljenih slik in v tretjem stolpcu število nepravilno razpoznanih testnih slik.

odstopanja od učnih podatkov. Primer je napačno označena Slika 4.5, kjer je klasifikator narobe prepoznal sliko kot BMW s 60 % gotovostjo, ko sta bila učna razreda le Lincoln in BMW. Če testiramo slike, ki ne vsebujejo nobenega izmed iskanih razredov, lahko kljub neujemanju razredov dobimo visoko gotovost, ker se je klasifikator priučil na šum v podatkih. Pri klasifikaciji manjšega števila razredov je priporočljivo dodati razred „vsi ostali“; s tem dodamo v učne podatke nekaj raznolikosti in se deloma izognemo prekomernemu sovpadanju z učnimi podatki.

Če število učnih primerov povečamo na 50 in ponovno pogledamo potek učenja, lahko iz Slike 4.6 vidimo, kako poteka učenje klasifikatorja. Uspešnost učenja desetih razredov se je spreminjala skozi iteraciji, klasifikator je primerjal različne značilke, ki bolje opišejo razrede. To lahko namiguje na veliko podobnost posameznih učnih primerov in ujemanje med razredi. Za zagotavljanje boljših rezultatov se klasifikator nauči drugih lastnosti in na njih preizkuša uspešnost klasificiranja. Izboljša se tudi število pravilno klasificiranih slik, kar je razvidno iz Tabele 4.2.

Ker v učenju razpoznavalnika ni večjih sprememb med manjšim številom razredov, smo pri naslednjem testiranju izbrali 200 učnih primerov in večje število razredov. S tem smo pridobili večjo gotovost klasifikatorja. Iz Ta-

Slika 4.5: Sliko avtomobila Lincoln smo obrezali in rotirali.

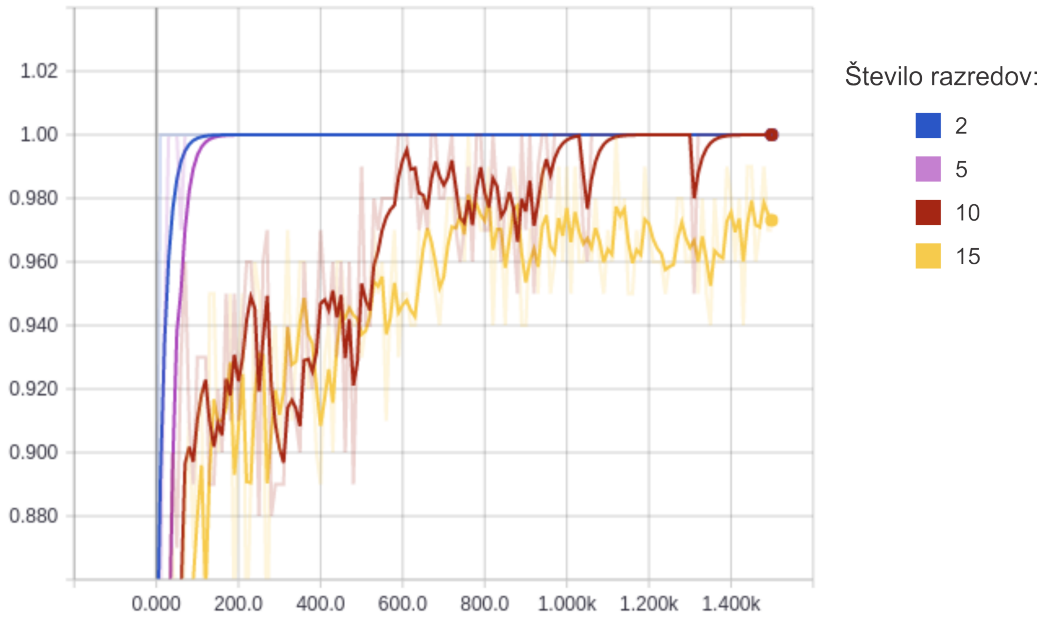
bele 4.3 je razvidno, koliko testnih slik izmed izbranega števila razredov je klasifikator razpoznal. V primeru napačno uvrščenih se gotovost klasifikatorja giblje v razponu 10–60 %, kar lahko zaobidemo z določitvijo meje, ki jo mora klasifikator doseči, da rezultat izpišemo. V primeru pravilno klasificiranega razreda je gotovost med 80 in 99 %. Slika 4.7 prikazuje matriko napačnega razvrščanja (angl. *confusion matrix*). Z rdečo barvo so prikazani najbolj številčni primeri, sledijo primeri krem barve in nato še sivine, ki označujejo 5 ali manj primerov. Največje število napačnega razvrščanja je v razredu Horch, v katerega so pogosto klasificirani tudi Packard, Rolls in Mercedes. Ko klasifikator učimo z majhno učno množico, se ta hitro prilagodi učnim podatkom in ni dovolj splošen. V primeru majhnega število razredov se klasifikator hitro nauči ločevati med njimi, vendar moramo za zanesljive rezultate zagotoviti zadostno število raznolikih učnih slik. Za manjše število razredov potrebujemo vsaj 200 slik, pri večjem številu razredov pa je lahko ta številka nekoliko manjša. Pri 15 razredih smo ob učenju zaznali spremembe že pri 30 učnih slikah. V primeru 10 razredov je bilo število učnih slik za variacije v učenju 50. Iz tega lahko sklepamo, da se učenje klasifikatorja začne šele pri 500 učnih slikah, vendar tudi to ne zagotavlja uspešnega ločevanja

Št. razredov / pripadnost	`Pravilno`	`Nepravilno`
2	153	15
5	318	39
10	724	74
15	765	117

Tabela 4.2: Primerjava uspešnosti klasifikatorja s 50 učnimi slikami. V prvem stolpcu je število učnih razredov, v drugem stolpcu je število pravilno opredeljenih slik in v tretjem stolpcu število nepravilno razpoznanih testnih slik.

Št. razredov / pripadnost	`Pravilno`	`Nepravilno`
5	315	63
10	557	94
15	732	129
22	765	117

Tabela 4.3: Primerjava uspešnosti klasifikatorja z 200 učnimi slikami.

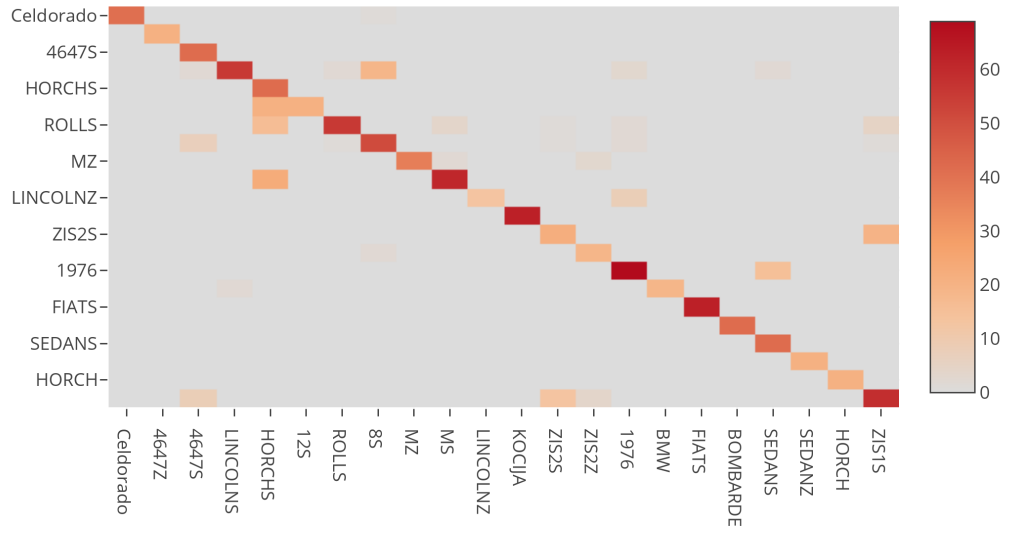


Slika 4.6: Spreminjanje gotovosti pri različnem številu razredov s po 50 učnimi slikami.

med razredi.

4.5 Umetno povečanje učne množice

Če želimo povečati število učnih podatkov in klasifikator bolj prilagoditi na učno množico, lahko to lahko dosežemo z obrezovanjem, rotacijo in s spreminjanjem svetlosti učnih slik. Primer obrezovanja slike z ohranjanjem motiva predstavlja Slika 4.8a, ki jo lahko še dodatno rotiramo. V našem primeru smo Sliko 4.8b rotirali za 18 stopinj. Če obrežemo vsako obstoječo sliko, to podvoji velikost množice. Če vsako sliko še rotiramo, to število učnih podatkov poveča za trikratnik. V primeru, da smo zajeli že ogromno različnih variacij okolja in osvetljenosti razredov in želimo klasifikator prilagoditi učnim podatkom ter poleg tega dodati malo variacije, lahko vsako sliko naključno rotiramo za –20 do 20 stopinj. Če želimo število primerov povečati, lahko



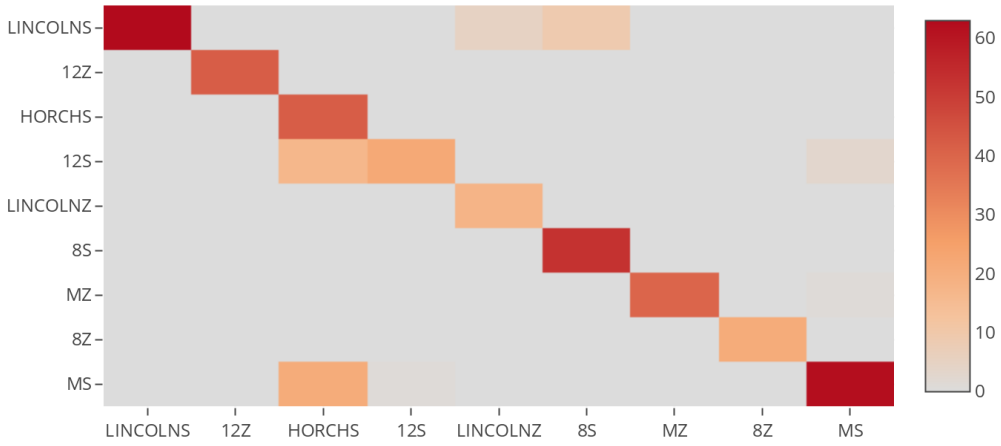
Slika 4.7: Slika prikazuje matriko zmot pri klasifikaciji 22 razredov.

naredimo več naključnih rotacij iste slike. S tem pokrijemo še nekaj dodatnih primerov, ki se zgodijo tudi pri uporabi. Tako podaljšamo učni proces, vendar posredno tudi povečamo gotovost klasifikatorja. S preizkušanjem smo naključno povečali učno množico za dva- do trikrat, kar je naneslo 700 do 1200 slik za posamezni učni razred. Učenje grafa je trajalo veliko dlje, vendar so rezultati boljši. Razpoznavalnik je uspešno ločil med 27 razredi z nekaj napakami v primerih, ki ne ležijo na diagonali; vidni so na Sliki 4.9. Zaradi lažje preglednosti je na grafu prikazan le izsek najpogostejših napak klasifikatorja, kjer ponovno rdeča barva predstavlja najštevilčnejše primere, sledijo ji manj številčni primeri, obarvani s krem barvo in sivi z najmanjšim številom zmot. Še vedno je najpogosteje napačno razpoznan razred Horch v primerih

.

(a) (b)

Slika 4.8: Primer manipulacije podatkov.



Slika 4.9: Slika prikazuje matriko zmot pri klasifikaciji 27 razredov.

Poglavje 5

Sklep

V delu smo razvili mobilno aplikacijo za razpoznavanje objektov na izbrani domeni. Za osnovo smo izbrali že vnaprej naučeno nevronsko mrežo Inception [29]. To smo naredili v okolju Tensorflow, ki omogoča enostavno učenje in optimizacijo. Spremljali smo proces učenja klasifikatorja in primerjali vpliv različnih parametrov na uspešnost razpoznavanja objektov. Dobljene rezultate smo izboljšali z umetnim povečanjem podatkovne zbirke in na podlagi tega dobili boljše rezultate. Priučeno nevronsko mrežo smo optimizirali z metodo kvantizacije. Z uporabo okolja Android Studio [2] smo generiran graf prenesli na mobilno napravo, kjer smo lahko tudi testirali uporabniško izkušnjo razpoznavanja.

5.1 Nadaljnje delo

Razpoznavanje objektov na slikah je področje, ki ima izjemen potencial in se hitro razvija. V zadnjem času se področje uporabe širi ravno zaradi mnogih sprememb in novosti. Možnost uporabe na mobilnih napravah prav tako odpira dodatne priložnosti za integracijo v različne aplikacije. Vključimo ga lahko tako v obogateno resničnost kot tudi v izobraževanja v šoli, naravi ali na odročnih lokacijah, saj za procesiranje zajetih slik ne potrebujemo povezave s strežnikom. Na ta način lahko tudi popestrimo izkušnjo obiska muzejev ter

naravnih ali kulturnih znamenitosti. Implementacija novejše arhitekture bi potencialno omogočala doseganje še boljših rezultatov in hitrejše razpoznavanje. Standardizacija mreže in uporaba v večjem številu aplikacijah pa bi omogočala manjše prostorske zahteve, saj se implementacije razpoznavalnika razlikujejo zgolj v zadnjih slojih.

Literatura

[1] S Agatonovic-Kustrin and R Beresford. Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research. *Journal of Pharmaceutical and Biomedical Analysis*, 22(5):717 – 727, 2000.

[2] Android studio. Dosegljivo: `https://developer.android.com/studio/index.html`, 2017. [Dostopano: 4. 9. 2017].

[3] Bag-of-words. Dosegljivo: `http://people.csail.mit.edu/torralba/shortCourseRLOC/index.html`, 2005. [Dostopano: 29. 8. 2017].

[4] C. M. Bishop. *Curvature-Driven Smoothing in Backpropagation Neural Networks*, pages 139–148. Springer London, London, 1992.

[5] Richard J. Bolton and David J. Hand. Statistical fraud detection: A review. *Statistical Science*, 17(3):235–249, 2002.

[6] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[7] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781, 2016.

[8] Cs231n: Convolutional Neural Networks for visual recognition. Dosegljivo: `https://cs231n.github.io/`, 2015. [Dostopano: 22. 8. 2017].

[9] Bruce Curry and Luiz Moutinho. Neural networks in marketing: Modelling consumer responses to advertising stimuli. *European Journal of Marketing*, 27(7):5–20, 1993.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[11] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[12] gzip. Dosegljivo: `http://www.gzip.org/`, 2003. [Dostopano: 30. 8. 2017].

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[14] ILSVRC. Dosegljivo: `http://www.image-net.org/challenges/LSVRC/`, 2009. [Dostopano: 22. 8. 2017].

[15] Tensorflow image retraining. Dosegljivo: `https://www.tensorflow.org/tutorials/image_retraining`, 2015. [Dostopano: 22. 8. 2017].

[16] Infer.NET. Dosegljivo: `http://infernet.azurewebsites.net/`, 2008. [Dostopano: 28. 8. 2017].

[17] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[18] Keras. Dosegljivo: `https://keras.io/`, 2017. [Dostopano: 22. 8. 2017].

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[20] MATLAB. Dosegljivo: `https://www.mathworks.com/products/matlab.html`, 1994. [Dostopano: 22. 8. 2017].

[21] PASCAL VOC. Dosegljivo: `http://host.robots.ox.ac.uk/pascal/VOC/`, 2005. [Dostopano: 30. 8. 2017].

[22] Tensorflow for poets. Dosegljivo: `https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0`, 2017. [Dostopano: 22. 8. 2017].

[23] Guangwu Qian and Lei Zhang. A simple feedforward convolutional conceptor neural network for classification. *Applied Soft Computing*, 2017.

[24] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[26] Leslie N. Smith and Nicholay Topin. Deep convolutional neural network design patterns. *CoRR*, abs/1611.00847, 2016.

[27] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[28] Christian Szegedy, Scott E. Reed, Dumitru Erhan, and Dragomir Anguelov. Scalable, high-quality object detection. *CoRR*, abs/1412.1441, 2014.

[29] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[30] Tensorflow. Dosegljivo: `https://www.tensorflow.org/`, 2017. [Dostopano: 15. 8. 2017].

[31] Tensorflow Optimize for Mobile. Dosegljivo: `https://codelabs.developers.google.com/codelabs/tensorflow-for-poets-2/#0`, 2016. [Dostopano: 22. 8. 2017].

[32] Theano. Dosegljivo: `http://www.deeplearning.net/software/theano/`, 2011. [Dostopano: 30. 8. 2017].

[33] Torch. Dosegljivo: `http://torch.ch/`, 2009. [Dostopano: 22. 8. 2017].

[34] A Vellido, P.J.G Lisboa, and J Vaughan. Neural networks in business: a survey of applications (1992–1998). *Expert Systems with Applications*, 17(1):51 – 70, 1999.

[35] Chong Wang and Kaiqi Huang. How to use bag-of-words model better for image classification. *Image and Vision Computing*, 38:65 – 74, 2015.

[36] Weka. Dosegljivo: `http://www.cs.waikato.ac.nz/ml/index.html`, 2016. [Dostopano: 22. 8. 2017].

[37] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke. The Microsoft 2017 Conversational Speech Recognition System. *ArXiv e-prints*, August 2017.

[38] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.